

# **White Paper Report**

Report ID: 98994

Application Number: HD5092710

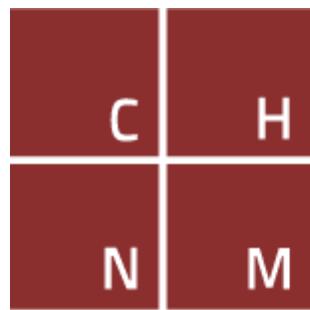
Project Director: Sharon Leon (sleon@gmu.edu)

Institution: George Mason University

Reporting Period: 7/1/2010-6/30/2011

Report Due: 9/30/2011

Date Submitted: 10/3/2011



**ROY ROSENZWEIG**  
Center FOR  
History AND  
New Media

## ***Scripto & Crowdsourcing Transcription:* a Technical Guide**

Sharon M. Leon & Jim Safley  
September, 2011

In July 2010, with support from the National Endowment for the Humanities and the National Historic Records and Publications Commission, the Roy Rosenzweig Center for History and New Media (CHNM) embarked upon an effort to build a generalized tool to facilitate the crowdsourcing of transcriptions from cultural heritage materials. The result of that work is *Scripto* (<http://scripto.org/>), a free, open source tool that enables content providers to crowdsource the transcription of their content, be it images, audio files, or video files. The tool offers a light-weight, easy to implement way for cultural heritage organizations and digital archives to engage their users with the work of transcription.

This guide provides a brief introduction to the use of crowdsourcing with cultural heritage collections and the technical philosophy underpinning *Scripto*'s development. Then, it details CHNM's work with the *Papers of the War Department* project as a case study for the implementation of *Scripto* with a large documentary archive. Finally, it offers potential users a guide to implementation and full documentation for the *Scripto* code.

## I. Why Crowdsourcing?

In proposing to open the transcription process up to crowdsourcing, CHNM drew upon the example and success of a host of other successful ventures. Though the goals and needs of digital archive and documentary editing projects are distinct from the goals of these crowdsourcing ventures, they did offer promising glimpses of the efficiencies and outcomes for our purposes. Each taps into the interests and passions of a segment of the public who contribute to the accumulated value and content of a project—whether software development or knowledge aggregation. The open source software movement—with successes like the Linux operating system and Mozilla's popular Firefox web browser—contains lessons for those of us interested in harnessing the expertise of a particular community to build a successful project. The developers who participate in open source software projects do so for their own reasons, but they contribute to the common good by applying their expertise to the demands and problems raised by software innovation. There are, however, several content-focused examples of crowdsourcing that are particularly revealing.

Wikipedia, the free online encyclopedia that is written and maintained by users around the world, is by far the most well-known instance of successful crowdsourcing. Launched in 2001, using a simple authoring and versioning software to manage articles, Wikipedia thrives on the contribution of thousands of anonymous users. The site currently receives roughly 400 million unique visitors each month and has more than 82,000 active contributors who create and edit over 19 million articles—more than 3.7 million of which are in English.<sup>1</sup> From its founding, teachers, parents and scholars have worried about the accuracy and content of Wikipedia, especially since articles from the free encyclopedia have long been the first result to appear in most search engines. Yet, the source of the occasional errors and incoherence of Wikipedia is also its power—the tremendously fast-moving contributions and collaboration of users means that errors are unlikely to remain for long before they are corrected. Similarly, the growth of articles and topics covered in the free encyclopedia reflects the energy and interests of the open source community of users. An assessment of the creation and revision history of articles on historical topics could provide a tremendous wealth of information about the concerns of the public with the past.<sup>2</sup>

In a similar, albeit much less extensive way, Flickr Commons <http://flickr.com/commons/> provides another example of tapping the collective knowledge of interested members of the public. Beginning with a seed contribution by the Library of Congress in January 2008, cultural institutions from around the world have contributed images and associated metadata to the Flickr Commons collection including the New York Public Library, the George Eastman museum, the Powerhouse Museum in Australia, and the Smithsonian Institution. The collections are freely available to the public, who can tag and comment on each image. A few years ago, the Library of Congress assessed their participation in the pilot project, reporting that over 2,500 Flickr users had left more than 7,000 comments on almost 3,000 images. Library staff culled important corrections and additions to captions, titles and the identification of individuals that were then incorporated back into the metadata of more than 500 items by August 2008. Based on these positive interactions with the public, the staff advocated that the library continue to draw upon public knowledge through the Flickr Commons project and other Web 2.0 interactive projects, declaring: “The benefits appear to far outweigh the costs and risks.” The Library of Congress’s success helped to encourage the Smithsonian Institution to join Flickr Commons, as a result, in the period between June and December 2008, their photographic contributions received over 625,000

---

<sup>1</sup>“About,” Wikipedia [http://en.wikipedia.org/wiki/About\\_Wikipedia](http://en.wikipedia.org/wiki/About_Wikipedia).

<sup>2</sup> Roy Rosenzweig, “Can History be Open Source? Wikipedia and the Future of the Past,” *The Journal of American History* 93:1 (June, 2006): 117-46, available at <http://chnm.gmu.edu/essays-on-history-new-media/essays/?essayid=42>.

views with many user comments and tags. Thus, trusted cultural heritage organizations are beginning to recognize the powerful ways that interested members of the public, scholars, and educators can contribute to the knowledge base related to collections.<sup>3</sup>

Outside of the United States, universities and national libraries have been running some very successful crowdsourcing projects. First, the National Library of Australia began running a project in August 2008 with members of the public correcting the results of optical character recognition software for their digitized newspapers. By project manager Rose Holley's estimates, there were roughly 6,000 participants who had corrected over 7 million lines of text by November 2009.<sup>4</sup> That project then became part of the Trove project <<http://trove.nla.gov.au/>>, which allows users not only to correct OCR, but also to contribute historic images, tag materials, and linking a host of other cultural materials. Currently, Trove provides users with access to nearly 250 million digital items. The results of these National Library of Australia projects show the remarkable range of contributions that public users can make to historical material, especially when they have the capacity to work across large collections.

University and public libraries have had good success with projects that ask users to participate in direct transcription of document images.<sup>5</sup> Of these, the University College London's Transcribe Bentham <<http://www.ucl.ac.uk/transcribe-bentham/>> project is probably the most well-known. In the course of its public work, the project has allowed for the transcription of close to 2,000 manuscripts from Jeremy Bentham's published and unpublished works. The transcriptions will form the foundation for future work on the *Collected Works of Jeremy Bentham*. MediaWiki is the system that underlies the Transcribe Bentham transcription work, and the project team has released the code for its MediaWiki plugins: <<http://code.google.com/p/tb-transcription-desk/>>. This code does not allow MediaWiki to interact with existing content management systems.

Finally, Zooniverse <<http://www.zooniverse.org/>> has supported a large number of "citizen science" projects in the last several years. Most focus on the crowdsourcing of big data, such as the Galaxy Zoo identification projects, but others have a more

---

<sup>3</sup> Michelle Springer, et. al., *For the Common Good: The Library of Congress Flickr Pilot Project, Final Report* (October 30, 2008): 36, available at <[http://www.loc.gov/rr/print/flickr\\_report\\_final.pdf](http://www.loc.gov/rr/print/flickr_report_final.pdf)>; Martin Kalfatovic, et al. "Smithsonian Team Flickr: a library, archives, and museums collaboration in web 2.0 space," *Archival Science* (October 2009), available at <<http://dx.doi.org/10.1007/s10502-009-9089-y>>.

<sup>4</sup> Rose Holley, "Crowdsourcing: How and Why Should Libraries Do It?" *D-Lib Magazine* 16:3/4 (March/April 2010), available at <<http://www.dlib.org/dlib/march10/holley/03holley.html>>.

<sup>5</sup> See also, Ben Brumfield's FromThePage <[http://beta.fromthepage.com/?ol=1\\_hd\\_logo](http://beta.fromthepage.com/?ol=1_hd_logo)>; New York Public Library's What's on the Menu? <<http://menus.nypl.org/>>; and the University of Iowa Libraries' Civil War Diaries and Letters Transcription Project <<http://digital.lib.uiowa.edu/cwd/transcripts.html>>.

historical focus. The Old Weather Project <http://www.oldweather.org/> is helping scientists recover weather observations included in the logbooks of Royal Navy vessels during the World War I era. The results of this work helps climate scientists build better models and provides historians with access to new data about the ships and their sailors. Similarly, the Ancient Lives Project <http://ancientlives.org/> has made the fragmentary Greek texts, the Ocyrhynchus Papyri, available for transcription. This work will facilitate the identification of known texts and the isolation of new ones, in turn contributing to a greater understanding of Greco-Roman Egypt. Zooniverse has released the code for their generalized transcription tool, Scribe <https://github.com/zooniverse/Scribe>.

All of these projects suggest that crowdsourcing transcription for digital collections can provide significant benefits to cultural heritage institutions. First, and foremost, public contributions provide transcriptions where there once were none, and where there likely would be none in the future. Second, in the case of documentary editing projects, staff can draw on the publicly contributed transcriptions to form a base for their editorial work. Editors may start with a rough transcription provided by interested users, and then apply the techniques and expertise of their training to produce a corrected transcription quickly. Moreover, allowing the public to contribute document transcriptions to digital collections has real-time benefits for the accessibility of the materials. Each transcription contributed to the archive is then made available to the collection management system's search engine. The result is ever-improving discoverability. The full text of the documents allows the search engines to surface the most relevant documents by better weighting their results.

Additionally, crowdsourcing transcriptions allow editors to better understand the ways in which some users interact with their archive. Necessarily the documents that are of the most interest to users will be transcribed most quickly and fully. Thus, the public contributions can serve as a barometer of the most interesting materials within a particular collection. This convergence of volunteer interest and the collection coverage points to perhaps the most important reason for launching collaborative work with the public: community building. As Trevor Owens has noted, the concept of crowdsourcing and the related turn towards gamification, can seem like an effort by projects to take advantage of public contributions simply as a free labor force. But, transcription volunteers make the contributions that they do because they find the work meaningful.<sup>6</sup> They are contributing to the usefulness of the collection and learning about history at the same time. This type of community building around collections increasing investment in cultural heritage and points to long term gains for the humanities.

## II. Technical Philosophy

---

<sup>6</sup> Trevor Owens, “Meanification and Crowdscrafting: Forget Badges,” *Playing the Past* (March 17, 2011), available at <http://www.playthepast.org/?p=1027>.

It's helpful to think of *Scripto* as a software library that mediates the communication between a content provider, a custom *Scripto* application, and MediaWiki <http://www.mediawiki.org/wiki/MediaWiki>. The content provider serves the content, usually a corpus of digital material that can be transcribed; the *Scripto* application juxtaposes that content (via a media viewer/player) with a transcription form; and MediaWiki serves as the transcription database, revision engine, and user administrator.

Unlike many of the recent systems that allow public users to contribute transcriptions, *Scripto* is designed specifically to work with existing content management systems, rather than to replace them with a second source repository. The rational behind this choice is two-fold. First, we firmly believe that cultural heritage institutions need to employ standardized metadata systems when they provide web content so that that content is as interoperable as possible. We did not want to create a tool that would impede that goal by forcing users to separate their source material from their metadata schema. Second, we wanted to offer the lowest barrier to use possible. Hence, the idea that users would have to duplicate their sources in a second system seemed like an unnecessary and unwise step.

We designed the *Scripto* library to be compatible with potentially any content provider. We accomplished this in two ways. First, the library defines an adapter interface that is used to establish two-way data mapping between the content provider and *Scripto*. Second, the library normalizes the content provider's identification scheme (no matter how informal and inconsistent) to enable fail-safe data transport between the content provider and MediaWiki.

*Scripto* is interface-agnostic, meaning that content providers are not tied to a single UI or a predetermined feature set. They have full control over how their transcription application looks and functions. This makes it possible to embed a fully customizable user interface into an existing context, such as a website or standalone application. It does require some technical proficiency to build a transcription application, but the provided API is straightforward and well documented.

We chose MediaWiki for the transcription database for several reasons: it is the most popular wiki application and has a sizable and active developer community; wiki markup is relatively easy to learn and there are useful editors available; it offers helpful features, such as discussion pages and user administration; and it comes with a powerful, fully-featured API that *Scripto* uses to interface with the content provider and transcription application.

Transcriptions are stored in the MediaWiki database until they are exported to the content provider's database by an administrator. This is done to ensure that transcriptions are complete and vetted before they are added to the content provider's

database. Each project should evaluate crowdsourced transcriptions according to its own criteria.

## III. Case Study: Papers of the War Department

The *Papers of the War Department, 1784-1800* (<http://wardepartmentpapers.org/>) > (PWD) provided the test case for our initial development and implementation of *Scripto*. PWD is powered by a custom, single-instance content management system, serving nearly 45,000 digitized manuscripts comprising over 80,000 page images. As such it provides an excellent example of a non-standard content provider, the type with which *Scripto* was designed to interface. The manuscripts were gathered and digitized piecemeal over a decade and followed peculiar organizational conventions that are reflected in its current data model. PWD launched in 2006, making digital images of the documents available for use by the public while the editorial staff worked to improve the metadata. In 2009, the basic lower resolution images were replaced by a set of higher resolution images. Then, in January 2011, we began internal testing of *Scripto* with archive. Finally, we opened the transcription tool to public volunteers in March 2011. The lessons learned from this case provide a valuable road map for those considering using *Scripto* with existing digital collections, or for those planning to build digital collections with crowdsourced transcription available from the start.

### A. Installation and customization

Our first step was to map PWD's idiosyncratic data model to *Scripto*, so we wrote a small class implementing *Scripto*'s adapter interface that responds to requests by the transcription service. We then linked the existing PWD website to a custom web application containing a document image viewer and input forms for transcription and discussion. We chose to use OpenLayers (<http://openlayers.org/>) as our image viewer because of its ability render high resolution image files directly in the web browser without investing in image conversion.

Our requirements were minimal so we chose a limited feature set. In addition to the image viewer and the transcription form, we included a discussion form, where transcribers can ask questions and clarify their work, and where administrators can answer questions and ask for clarification. Since PWD's transcription application requires users to register and log in, transcribers can view a list of document pages to which they have contributed. This makes it easy for users to return to their previous work. Administrators have the authority to protect pages from further edits and to export the document transcription to the PWD database.

Beyond the core operational features of the system, we did some work to assure that

the tool meshed well with the needs of the PWD archive. Thus, our designer created a number of mock-ups for the transcription interface. Using both paper prototypes and unstyled builds of the interface, we did user testing with PWD editors, CHNM graduate research assistants who had worked with the PWD archive, and with several individuals who had no familiarity with the system. As a result of this user testing, we fundamentally reoriented the transcription interface, rejecting the common side-by-side document and transcription window positioning in favor of a top and bottom orientation. By positioning the OpenLayers image viewer on top with the transcription window below, we maximized the width of the viewer window. This decision dramatically increased the efficiency of transcribers by allowing them to view more of single line of text while zooming in on an image. Next, with this orientation, we narrowed the width of the transcription window to make it comfortable for typed text. Finally, we positioned the list of document page images to the left of the window, allowing a volunteer to easily proceed through multipage documents.

## B. Launch and oversight

In preparation for launching *Scripto* with PWD, we created a registration workflow for new volunteers. Although MediaWiki can be configured to allow users to create their own accounts or to edit documents without being logged into the system, we felt strongly that it was important to maintain editorial control over the transcription system through user accounts and logins. Thus, we configured MediaWiki to prohibit document edits by anonymous users, and placed the process of account creation in the hands of the PWD editorial team. To manage that process, we created a Google Form to gather registration information from volunteer transcribers. That form requires the minimal data for account creation (username and email address), but requests a full name, affiliation, country, zip code, and the reason the user is interested in working with the PWD archive. The results of these form submissions are gathered in a Google spreadsheet, which a PWD editor uses to hand-create MediaWiki accounts for each user. Once the user has verified the account by setting her password, she is set to begin transcription work.

In addition to the basic transcription interface of the document viewer and the transcription window, the PWD implementation of *Scripto* required us to create a number of support structures. We developed a set of static text pages that offered potential contributors a clear and concise introduction to the transcription project and its role in the larger PWD project. These included both the invitation to participate as transcribers and the short guidelines for creating good transcriptions within the conventions of the *Scripto* system. Also, we provided contributors with a large list of potential candidates for transcription. While many of our participants were drawn to the project by their own research interests and had clear ideas about the documents they wanted to work with, we realized that we were likely to attract many volunteers who simply wanted to aid the progress of the project and who would welcome our

suggestions to direct their work. Thus, PWD assistant editors continually nominate documents for transcription from their ongoing work with the collection. Currently, there are roughly 200 nominated documents.

To launch the transcription venture, we coordinated a publicity campaign that included blog posts, twitter coverage, and direct messages to listservs with high traffic from early Americanists, those teaching the US history survey, and a full range of genealogical organizations. Six months into the launch of *Scripto* with PWD, we have seen a tremendous response from the public. In total, we have over 300 user accounts, with roughly 70 that are regularly active in submitting transcriptions. Together the contributors have rendered 446 complete document transcriptions, ranging from single page memos to complex multipage documents. Additionally, there are nearly 200 documents currently in process.

Each day a PWD editor spends some time working with the volunteers and the transcription submissions. First, he monitors the registration list and creates new accounts. Next he surveys all of the newly created page transcriptions, making some corrections and edits. Then, he reads the discussion pages associated with the transcriptions. Over the past six months, he has engaged in 99 conversations with transcription volunteers about their work. Finally, he spends some time blogging and tweeting about the nominated documents, the completed transcriptions, and other project progress.

## IV. Guide to Implementation

With the example of the PWD case study, other cultural heritage institutions may be ready to implement *Scripto* with their own collections. The *Scripto* team is hard at work creating adaptors for some of the most common content management systems (Omeka, Drupal, WordPress, etc), but the following guide offers guidance for the programmer who wants to manage her own implementation.

### A. Features

To a large extent, *Scripto*'s features parallel those provided by MediaWiki:

- Authenticate users and control user access
- Transcribe and discuss pages
- Protect and unprotect pages
- Watch and unwatch pages
- View all recent changes
- View user watchlist

- View recent user contributions
- View page revision history
- Revert to previous revision
- View differences between page revisions
- Parse transcriptions (wikitext, HTML, plain text)
- Export transcription to external database

## B. Software Requirements

- PHP 5.2.4+
- Zend Framework 1.10+
- MediaWiki 1.15.4+

## C. Installation

The basic installation and customization of *Scripto* requires five key steps:

1. Download and install MediaWiki: <http://www.mediawiki.org/wiki/MediaWiki>
2. Download the Zend Framework library: <http://framework.zend.com/>
3. Download the *Scripto* library: <https://github.com/chnm/Scripto>
4. Write a custom adapter to the content provider, implementing [\*Scripto\\_Adapter\\_Interface\*](#). The *Scripto* library comes with a detailed example of how to implement this interface (see *Scripto\_Adapter\_Example*).
5. Use the *Scripto* library API to build a transcription application

It's important to note that the *Scripto* library does not include a user interface, but it does provide an API that developers can use to build a custom UI that conforms to their particular functional and stylistic requirements. The *Scripto* application API is found in class *Scripto* and the document API is found in class *Scripto\_Document*. Consider all public members of these classes part of the API.

We built the *Scripto* library to accommodate many features that are native to MediaWiki, such as user authentication, revision history, recent changes, watchlist, page discussion, and page protection. Developers should design a custom interface with only the features they need, including a media viewer/player of their choice.

There are many media viewer/players available for use. The user interface could simply display a flat image alongside the transcription form, but it may need special features for large images and audio/video files. OpenLayers is an excellent open source solution for an image viewer with pan and zoom capabilities. Microsoft's Zoom.it and the Google Docs Viewer offer easy to use web APIs for elegant image and document viewing. Audio and video media players are, unfortunately, not as prevalent or configurable.



## Appendix A: Glossary

- **content provider:** a provider of digital content and metadata about that content.
- **document:** *Scripto*'s name for a set of document pages representing a single resource. A document may have one or more document pages.
- **document page:** *Scripto*'s name for any digital file that can be transcribed, usually an image, audio, or video file.
- **MediaWiki:** a popular wiki software application that *Scripto* uses as its transcription database, revision engine, and user administrator.
- **Scripto adapter interface:** a design pattern used by *Scripto*, implemented by the content provider to translate its data model (at runtime) into one compatible with *Scripto*.
- **Scripto API:** the application programming interface used to build a *Scripto* application.
- **Scripto application:** a custom user interface combining a transcription form, a media viewer/player, and possibly other features available through the *Scripto* API.
- **Scripto library:** the *Scripto* codebase; a set of classes containing the *Scripto* API.

## Appendix B: Code Documentation

All *Scripto* code is available to developers for use and modification under the GPL 3.0 license: <https://github.com/chnm/Scripto>.

### Scripto Classes

#### Class Scripto

Represents a Scripto application. Serves as a connector object between the external system API and MediaWiki API.

protected Scripto::\$\_adapter

- Var Scripto\_Adapter\_Interface
- The adapter object for the external system.

protected Scripto::\$\_mediawiki

- Var Scripto\_Service\_MediaWiki
- The MediaWiki service object.

protected Scripto::\$\_userInfo

- Var array
- Cached information about the current user.

Constructor void public function Scripto::\_\_construct(\$adapter, \$mediawiki)

- Function Parameters:
  - Scripto\_Adapter\_Interface \$adapter The adapter object.
  - array|Scripto\_Service\_MediaWiki \$mediawiki If an array:
    - \$mediawiki['api\_url']: required; the MediaWiki API URL
    - \$mediawiki['db\_name']: required; the MediaWiki database name
    - \$mediawiki['pass\_cookies']: optional pass cookies to the web browser via API client
- Construct the Scripto object.

void public function Scripto::\_\_call(\$pageId)

- Function Parameters
  - string|null \$pageId The unique page identifier.
- Sets the current page ID, the base title used by MediaWiki, and information about the MediaWiki transcription and talk pages.

bool public function Scripto::documentExists(\$id)

- Function Parameters

- string/int \$id The unique document identifier.
- Check whether the specified document exists in the external system.
- Uses Scripto\_Adapter\_Interface::documentExists()

`Scripto_Document public function Scripto::getDocument($id)`

- Function Parameters
  - string/int \$id The unique document identifier.
- Get a Scripto\_Document object.
- See Scripto\_Document

`void public function Scripto::login($username, $password)`

- Function Parameters
  - string \$username The MediaWiki user's username.
  - string \$password The MediaWiki user's password.
- Login via the MediaWiki service.

It is possible to restrict account creation in MediaWiki.

- Link [http://www.mediawiki.org/wiki/Manual:Preventing\\_access#Restrict\\_account\\_creation](http://www.mediawiki.org/wiki/Manual:Preventing_access#Restrict_account_creation)
- Uses Scripto\_Service\_MediaWiki::login()

`void public function Scripto::logout()`

- Logout via the MediaWiki service.
- Uses Scripto\_Service\_MediaWiki::logout()

`bool public function Scripto::canExport([$groups = array('sysop', 'bureaucrat')])`

- Function Parameters:
  - array \$groups The MediaWiki groups allowed to export.
- Determine if the current user can export transcriptions to the external system.

`bool public function Scripto::canProtect()`

- Determine if the current user can protect MediaWiki pages.

`void public function Scripto::setUserInfo()`

- Set the current user's information.

Under normal circumstances calling this method directly is unnecessary, but is helpful when authenticating after construction and when a login is not called, like when hijacking cookies for command line authentication.

- Uses Scripto\_Service\_MediaWiki::getUserInfo()

```
string public function Scripto::getUserName()
```

- Return the name of the current user.

```
array public function Scripto::getUserDocumentPages([$limit = 10])
```

- Function Parameters:
  - int \$limit The number of document pages to return.
- Get the current user's most recently contributed document pages.
- Uses Scripto\_Service\_MediaWiki::getUserContributions()

```
array public function Scripto::getRecentChanges([$limit = 10])
```

- Function Parameters:
  - int \$limit The number of recent changes to return.
- Get the recent changes.
- Uses Scripto\_Service\_MediaWiki::getRecentChanges()
- Link [http://www.mediawiki.org/wiki/Manual:Namespace#Built-in\\_namespaces](http://www.mediawiki.org/wiki/Manual:Namespace#Built-in_namespaces)

```
array public function Scripto::getWatchlist([$limit = 10])
```

- Function Parameters:
  - int \$limit int \$limit The number of recent changes to return.
- Uses Scripto\_Service\_MediaWiki::getWatchlist()
- Link <http://www.mediawiki.org/wiki/API:Watchlist>

```
array public function Scripto::getAllDocuments()
```

- Get all documents from MediaWiki that have at least one page with text.
- Uses Scripto\_Service\_MediaWiki::getAllPages()
- Return An array following this format: array({document ID} => array(['mediawiki\_titles'] => array({page ID} => {mediawiki title}, {...}), ['document\_title'] => {document title}), {...})

```
string public function Scripto::getRevisionDiff($fromRevisionId, [$toRevisionId = 'prev'])
```

- Function Parameters:
  - int \$fromRevisionId The revision ID from which to diff.
  - int|string \$toRevisionId The revision to which to diff. Use the revision ID, "prev", "next", or "cur".
- Get the difference between two page revisions.
- Uses Scripto\_Service\_MediaWiki::getRevisionDiff()
- Return An HTML table without the wrapping <table> tag containing difference markup, pre-formatted by MediaWiki. It is the responsibility of implementers to wrap the result with table tags.

```
array public function Scripto::getRevision($revisionId)
```

- Function Parameters:
  - int \$revisionId The ID of the page revision.
- Get properties of the specified page revision.
- Uses Scripto\_Service\_MediaWiki::getRevisions()

```
string static public function Scripto::getChangeAction([$hints = array()])
```

- Function Parameters:
  - array \$hints Keyed hints from which to infer an change action:
    - comment
    - log\_action
    - revision\_id
- Infer a change action verb from hints contained in various responses.

```
bool static public function Scripto::isValidApiUrl($apiUrl)
```

- Function Parameters:
  - string \$apiUrl The MediaWiki API URL to validate.
- Determine whether the provided MediaWiki API URL is valid.
- Uses Scripto\_Service\_MediaWiki::isValidApiUrl()

```
string static public function Scripto::removeHtmlAttributes($html)
```

- Function Parameters:
  - string \$html
- Remove all HTML attributes from the provided markup.

This filter is useful after getting HTML from the MediaWiki API, which often contains MediaWiki-specific attributes that may conflict with local settings.

## **Class Scripto\_Adapter\_Exception**

### **Class Scripto\_Adapter\_Interface**

Interface for Scripto adapter classes.

```
bool public function Scripto_Adapter_Interface::documentExists($documentId)
```

- Function Parameters:
  - int|string \$documentId The unique document ID
- Indicate whether the document exists in the external system. Implementers must provide a unique identifier for every document. We highly recommend using

unique keys from the external database whenever possible (e.g. the document ID).

- Return True: it exists; false: it does not exist

```
bool public function Scripto_Adapter_Interface::documentPageExists($documentId,
$pagId)
```

- Function Parameters:
  - \$documentId The unique document ID
  - \$pagId The unique page ID
- Indicate whether the document page exists in the external system. Implementers must provide a unique identifier for every document page per document. We highly recommend using unique keys from the external database whenever possible (e.g. the page ID).
- Return True: it exists; false: it does not exist

```
bool public function
Scripto_Adapter_Interface::documentPageTranscriptionIsImported($documentId,
$pagId)
```

- Function Parameters:
  - \$documentId The document ID
  - \$pagId The page ID
- Indicate whether the document page transcription has been imported.
- Usedby Scripto\_Document::isExportedPage()
- Return True: has been imported; false: has not been imported

```
bool public function
Scripto_Adapter_Interface::documentTranscriptionIsImported($documentId)
```

- Function Parameters:
  - int|string \$documentId The document ID
- Indicate whether the document transcription has been imported.
- Usedby Scripto\_Document::isExported()
- Return True: has been imported; false: has not been imported

```
int|string public function
Scripto_Adapter_Interface::getDocumentFirstPagId($documentId)
```

- Function Parameters:
  - int|string \$documentId The document ID
- Get the first page of the document.
- Usedby Scripto\_Document::getFirstPagId()

```
string public function
Scripto_Adapter_Interface::getDocumentPageFileUrl($documentId, $pagId)
```

- Function Parameters:
  - int|string \$documentId The unique document ID int|string \$pageId The unique page ID
- Get the URL of the specified document page file.
- Usedby Scripto\_Document::getPageFileUrl()

```
string public function Scripto_Adapter_Interface::getDocumentPageName($documentId,
$pageId)
```

- Function Parameters:
  - \$documentId The document ID
  - \$pageId The unique page ID
- Get the name of the document page.

```
array public function Scripto_Adapter_Interface::getDocumentPages($documentId)
```

- Function Parameters:
  - int|string \$documentId The unique document ID
- Get all the pages belonging to the document. Implementers must provide a unique identifier for every page per document. These IDs must have corresponding page names, and must be in sequential page order. Page IDs must be unique but do not have to be in natural order. Page names do not have to be unique. For the page IDs we highly recommend using unique keys from the external database whenever possible (e.g. the file ID).

This page ID will be used to query the adapter for page file URLs, so they must be no ambiguity. The return value must follow this format: array([pageId] => [pageName], [...])

Example return values: array(2011 => 'Title Page', 1999 => 'Page 1', 4345 => 'Page 2') array('page\_1' => 1, 'page\_2' => 2, 'page\_3' => 3)

- Usedby Scripto\_Document::getPages()
- Return An array containing page identifiers as keys and page names as values, in sequential page order.

```
string public function Scripto_Adapter_Interface::getDocumentTitle($documentId)
```

- Function Parameters:
  - int|string \$documentId The document ID
- Get the title of the document.

```
bool public function
```

```
Scripto_Adapter_Interface::importDocumentPageTranscription($documentId, $pageId,  
$text)
```

- Function Parameters:
  - \$documentId The document ID
  - \$pageId The page ID
  - string \$text The text to import
- Import a document page's transcription into the external system.
- Used by Scripto\_Document::exportPage()
- Return True: success; false: fail

```
bool public function
```

```
Scripto_Adapter_Interface::importDocumentTranscription($documentId, $text)
```

- Function Parameters:
  - int|string \$documentId The document ID
  - string \$text The text to import
- Import an entire document's transcription into the external system.
- Used by Scripto\_Document::export()
- Return True: success; false: fail

## Class Scripto\_Document

Represents a Scripto document. Serves as a connector object between the external system API and MediaWiki API.

```
Scripto_Document::BASE_TITLE_DELIMITER = ''
```

- The delimiter used to separate the document and page IDs in the base title.

```
Scripto_Document::BASE_TITLE_PREFIX = ''
```

- The prefix used in the base title to keep MediaWiki from capitalizing the first character.

```
Scripto_Document::TITLE_BYTE_LIMIT = 256
```

- The maximum bytes MediaWiki allows for a page title.

```
protected Scripto_Document::$_adapter
```

- Var Scripto\_Adapter\_Interface
- The adapter object for the external system.

```
protected Scripto_Document::$_baseTitle
```

- Var string

- The base title (i.e. without a namespace) of the corresponding MediaWiki page.

`protected Scripto_Document::$_id`

- Var string
- The document ID provided by the external system.

`protected Scripto_Document::$_mediawiki`

- Var `Scripto_Service_MediaWiki`
- The MediaWiki service object.

`protected Scripto_Document::$_pageId`

- Var string
- The document page ID provided by the external system.

`protected Scripto_Document::$_pageName`

- Var string
- The document page name provided by the external system.

`protected Scripto_Document::$_talkPageInfo`

- Var array
- Information about the current talk page.

`protected Scripto_Document::$_title`

- Var string
- The document title provided by the external system.

`protected Scripto_Document::$_transcriptionPageInfo`

- Var array
- Information about the current transcription page.

`Constructor void public function Scripto_Document::__construct($id, $adapter, $mediawiki)`

- Function Parameters:
  - string|int `$id` The unique document identifier.
  - `Scripto_Adapter_Interface` `$adapter` The adapter object.
  - array|`Scripto_Service_MediaWiki` `$mediawiki` `Scripto::mediawikiFactory()`
- Construct the Scripto document object.

`string static public function Scripto_Document::base64UrlDecode($str)`

- Function Parameters:

- string \$str
- Decode a string from a URL-safe Base64.

`string static public function Scripto_Document::base64UrlEncode($str)`

- Function Parameters:
  - string \$str
- Encode a string to URL-safe Base64.
- Link [http://en.wikipedia.org/wiki/Base64#URL\\_applications](http://en.wikipedia.org/wiki/Base64#URL_applications)

`bool public function Scripto_Document::canEditTalkPage()`

- Determine if the current user can edit the MediaWiki talk page.

`bool public function Scripto_Document::canEditTranscriptionPage()`

- Determine if the current user can edit the MediaWiki transcription page.

`array static public function Scripto_Document::decodeBaseTitle($baseTitle)`

- Function Parameters:
  - string|int \$baseTitle
- Decode the base title.
- Return An array containing the document ID and page ID

`void public function Scripto_Document::editTalkPage($text)`

- Function Parameters:
  - string \$text The wikitext of the transcription.
- Edit the MediaWiki talk page for the current document.
- Uses `Scripto_Service_MediaWiki::edit()`

`void public function Scripto_Document::editTranscriptionPage($text)`

- Function Parameters:
  - string \$text The wikitext of the transcription.
- Edit the MediaWiki transcription page for the current document.
- Uses `Scripto_Service_MediaWiki::edit()`

`string static public function Scripto_Document::encodeBaseTitle($documentId, $pageId)`

- Function Parameters:
  - \$documentId The document ID
  - \$pageId The page ID
- Encode a base title that enables fail-safe document page transport between the external system, Scripto, and MediaWiki. The base title is the base MediaWiki page title that corresponds to the document page. Encoding is necessary to allow

all Unicode characters in document and page IDs, even those not allowed in URL syntax and MediaWiki naming conventions. Encoding in Base64 allows the title to be decoded.

The base title has four parts:

1. A title prefix to keep MediaWiki from capitalizing the first character
2. A URL-safe Base64 encoded document ID
3. A delimiter between the encoded document ID and page ID
4. A URL-safe Base64 encoded page ID

- Link [http://en.wikipedia.org/wiki/Base64#URL\\_applications](http://en.wikipedia.org/wiki/Base64#URL_applications)
- Link [http://en.wikipedia.org/wiki/Wikipedia:Naming\\_conventions\\_%28technical\\_restrictions%29](http://en.wikipedia.org/wiki/Wikipedia:Naming_conventions_%28technical_restrictions%29)
- Return The encoded base title

```
void public function Scripto_Document::export([$type = 'plain_text'], [$pageDelimiter = "\n"])
```

- Function Parameters:
  - string \$type The type of text to set, valid options are plain\_text, html, and wikitext.
  - string \$pageDelimiter The delimiter used to stitch pages together.
- Export the entire document transcription to the external system by calling the adapter.
- Uses Scripto\_Adapter\_Interface::importDocumentTranscription()

```
void public function Scripto_Document::exportPage([$type = 'plain_text'])
```

- Function Parameters:
  - string \$type The type of text to set, valid options are plain\_text, html, and wikitext.
- Export the document page transcription to the external system by calling the adapter.
- Uses Scripto\_Adapter\_Interface::importDocumentPageTranscription()

```
string public function Scripto_Document::getBaseTitle()
```

- Get this document's current base title.

```
array public function Scripto_Document::getFirstPageId()
```

- Get this document's first page ID from the adapter.
- Uses Scripto\_Adapter\_Interface::getDocumentFirstPageId()

```
string|int public function Scripto_Document::getId()
```

- Get this document's ID.

```
string public function Scripto_Document::getPageFileUrl()
```

- Get this document's current page file URL from the adapter.
- Uses Scripto\_Adapter\_Interface::getDocumentPageFileUrl()

```
string|int public function Scripto_Document::getPageId()
```

- Get this document's current page ID.

```
void public function Scripto_Document::getPageName()
```

- Get this document page's name.

```
array public function Scripto_Document::getPages()
```

- Get all of this document's pages from the adapter.
- Uses Scripto\_Adapter\_Interface::getDocumentPages()

```
array public function Scripto_Document::getTalkPageHistory([$limit = 10],  
[$startRevisionId = null])
```

- Function Parameters:
  - int \$limit The number of revisions to return.
  - int \$startRevisionId The revision ID from which to start.
- Get the MediaWiki talk page revision history for the current page.

```
string public function Scripto_Document::getTalkPageHtml()
```

- Get the MediaWiki talk page HTML for the current page.
- Uses Scripto\_Service\_MediaWiki::getLatestRevisionHtml()

```
array public function Scripto_Document::getTalkPageInfo()
```

- Get information about the current MediaWiki talk page.

```
string public function Scripto_Document::getTalkPageMediawikiUrl()
```

- Get the MediaWiki URL for the current talk page.

```
string public function Scripto_Document::getTalkPagePlainText()
```

- Get the MediaWiki talk plain text for the current page.
- Uses Scripto\_Service\_MediaWiki::getLatestRevisionHtml()

```
string public function Scripto_Document::getTalkPageWikitext()
```

- Get the MediaWiki talk page wikitext for the current page.

- Uses Scripto\_Service\_MediaWiki::getLatestRevisionWikitext()

void public function Scripto\_Document::getTitle()

- Get this document's title.

array public function Scripto\_Document::getTranscriptionPageHistory([\$limit = 10],  
[\$startRevisionId = null])

- Function Parameters:
  - int \$limit The number of revisions to return.
  - int \$startRevisionId The revision ID from which to start.
- Get the MediaWiki transcription page revision history for the current page.

string public function Scripto\_Document::getTranscriptionPageHtml()

- Get the MediaWiki transcription page HTML for the current page.
- Uses Scripto\_Service\_MediaWiki::getLatestRevisionHtml()

array public function Scripto\_Document::getTranscriptionPageInfo()

- Get information about the current MediaWiki transcription page.

string public function Scripto\_Document::getTranscriptionPageMediawikiUrl()

- Get the MediaWiki URL for the current transcription page.

string public function Scripto\_Document::getTranscriptionPagePlainText()

- Get the MediaWiki transcription page plain text for the current page.
- Uses Scripto\_Service\_MediaWiki::getLatestRevisionHtml()

string public function Scripto\_Document::getTranscriptionPageWikitext()

- Get the MediaWiki transcription page wikitext for the current page.
- Uses Scripto\_Service\_MediaWiki::getLatestRevisionWikitext()

bool public function Scripto\_Document::isExported()

- Determine whether all of this document's transcription pages were already exported to the external system.
- Uses Scripto\_Adapter\_Interface::documentTranscriptionIsImported()

bool public function Scripto\_Document::isExportedPage()

- Determine whether the current transcription page was already exported to the external system.
- Uses Scripto\_Adapter\_Interface::documentPageTranscriptionIsImported()

`bool public function Scripto_Document::isProtectedTalkPage()`

- Determine whether the current talk page is edit protected.

`bool public function Scripto_Document::isProtectedTranscriptionPage()`

- Determine whether the current transcription page is edit protected.

`bool public function Scripto_Document::isWatchedPage()`

- Determine whether the current user is watching the current page.

`void public function Scripto_Document::protectTalkPage()`

- Protect the current talk page.

`void public function Scripto_Document::protectTranscriptionPage()`

- Protect the current transcription page.

`void public function Scripto_Document::setPage($pageId)`

- Function Parameters:
  - `string|null $pageId` The unique page identifier.
- Set the current document page. Sets the current page ID, the base title used by MediaWiki, and information about the MediaWiki transcription and talk pages.

`void public function Scripto_Document::unprotectTalkPage()`

- Unprotect the current talk page.

`void public function Scripto_Document::unprotectTranscriptionPage()`

- Unprotect the current transcription page.

`void public function Scripto_Document::unwatchPage()`

- Unwatch the current page. Unwatching a transcription page implies unwatching its talk page.
- Uses `Scripto_Service_MediaWiki::watch()`

`void public function Scripto_Document::watchPage()`

- Watch the current page. Watching a transcription page implies watching its talk page.
- Uses `Scripto_Service_MediaWiki::watch()`

`bool protected function Scripto_Document::_canEdit($pageProtections)`

- Function Parameters:
  - `array $pageProtections`

- Determine if the current user can edit the specified MediaWiki page.
- Uses Scripto\_Service\_MediaWiki::getUserInfo()

```
array protected function Scripto_Document::_getPageHistory($title, [$limit = 10],  
[$startRevisionId = null])
```

- Function Parameters:
  - string \$title
  - int \$limit
  - int \$startRevisionId
- Get the revisions for the specified page.
- Uses Scripto\_Service\_MediaWiki::getRevisions()

```
array protected function Scripto_Document::_getPageInfo($title)
```

- Function Parameters:
  - string \$title
- Get information for the specified page.
- Uses Scripto\_Service\_MediaWiki:: getInfo()

```
string protected function Scripto_Document::_getPageMediawikiUrl($title)
```

- Function Parameters:
  - string \$title
- Get the MediaWiki URL for the specified page.
- Uses Scripto\_Service\_MediaWiki::getSiteInfo()

```
bool protected function Scripto_Document::_isProtectedPage($pageProtections)
```

- Function Parameters:
  - array \$pageProtections The page protections from the page info: Scripto\_Document::\$\_transcriptionPageInfo or Scripto\_Document::\$\_talkPageInfo.
- Determine whether the provided protections contain an edit protection.

```
void protected function Scripto_Document::_protectPage($title, $protectToken)
```

- Function Parameters:
  - string \$title
  - string \$protectToken
- Protect the specified page.
- Uses Scripto\_Service\_MediaWiki::protect()

```
void protected function Scripto_Document::_unprotectPage($title, $protectToken)
```

- Function Parameters:

- string \$title
- string \$protectToken
- Unprotect the specified page.
- Uses Scripto\_Service\_MediaWiki::protect()

## **Class Scripto\_Exception**

### **Class Scripto\_Service\_Exception**

Represents a Scripto\_Service exception.

### **Class Scripto\_Service\_MediaWiki**

MediaWiki API client.

Scripto\_Service\_MediaWiki::COOKIE\_PREFIX = 'scripto\_'

- The cookie name prefix, used to namespace Scripto/MediaWiki cookies when passed protected to the browser.

protected Scripto\_Service\_MediaWiki::\$\_actions

- Var array(
   
 'parse' => array( 'text', 'title', 'page', 'prop', 'pst', 'uselang'),
 'edit'=>array('title','section','text','token','summary','minor','notminor','bot','basetime',
 'stamp','starttime','recycle','createonly','nocreate','watchlist','md5','captcha',
 'captchaword','undo','undoafter'),
 'protect'=>array('title','token','protections','expiry','reason','cascade'),
 'watch'=>array('title','unwatch'),
 'query'=>array(
 // title specifications
 'titles','revids','pageids',
 // submodules
 'meta','prop','list',
 // meta submodule
 'siprop','sifilteriw','sishowalldb','sinumberinggroup','uiprop',
 // prop
 submodule 'inprop','intoken','indexpageids','incontinue','rvprop','rvcontinue','rvlimit',
 'rvstartid','rvendid','rvstart','rvend','rvdir','rvuser','rvexcludeuser','rvexpandtemplates','rvgeneratexml','rvsection','rvtoken','rvdiffsto','rvdifftotext',
 // list
 submodule 'ucprop','ucuser','ucuserprefix','ucstart','ucend','uccontinue','ucdir','ucli

```
mit', 'ucnamespace', 'ucshow', 'rcprop', 'rcstart', 'rcend', 'rmdir', 'rclimit', 'rcnamespace', 'rcuser', 'rcexcludeuser', 'rctype', 'rcshow', 'wlprop', 'wlstart', 'wlend', 'wldir', 'wllimit', 'wlname', 'wluser', 'wlexcludeuser', 'wlowner', 'wltoken', 'wlallrev', 'wlshow', 'aplimit', 'apminsize', 'apmaxsize', 'apprefix', 'apfrom', 'apnamespace', 'apfilterredir', 'apfilterlanglinks', 'apptype', 'applevel', 'apdir', ),  
'login'=>array('lgname', 'lgpassword', 'lgtoken'),  
'logout'=>array())
```

- Valid MediaWiki API actions and their valid parameters. This whitelist is used to prohibit invalid actions and parameters from being set to API requests.

protected Scripto\_Service\_MediaWiki::\$\_cookieSuffixes

- Var array('\_session', 'UserID', 'UserName', 'Token')
- Scripto/MediaWiki cookie name suffixes.

protected Scripto\_Service\_MediaWiki::\$\_dbName

- Var string
- The MediaWiki database name, used to namespace Scripto/MediaWiki cookies.

protected Scripto\_Service\_MediaWiki::\$\_passCookies

- Var bool
- Pass Scripto cookies to the web browser

Constructor void public function Scripto\_Service\_MediaWiki::\_\_construct(\$apiUrl, \$dbName, [\$passCookies = true])

- Function Parameters:
  - string \$apiUrl The URL to the MediaWiki API.
  - string \$dbName The name of the MediaWiki database.
  - bool \$passCookies Pass cookies to the web browser.
- Constructs the MediaWiki API client.
- Link [http://www.mediawiki.org/wiki/API:Main\\_page](http://www.mediawiki.org/wiki/API:Main_page)

array public function Scripto\_Service\_MediaWiki::edit(\$title, \$text, [\$edittoken = null], [\$params = array()])

- Function Parameters:
  - string \$title
  - string \$text
  - string|null \$edittoken
  - array \$params
- Create or edit a given page.
- Link <http://www.mediawiki.org/wiki/>

[Manual:Preventing\\_access#Restrict\\_editing\\_of\\_all\\_pages](#)

- Link <http://www.mediawiki.org/wiki/API>Edit>
- Usedby `Scripto_Document::editTalkPage()`
- Usedby `Scripto_Document::editTranscriptionPage()`

`array public function Scripto_Service_MediaWiki::getAllPages([$params = array()])`

- Function Parameters:
  - `array $params`
- Gets a list of pages.
- Link <http://www.mediawiki.org/wiki/API>Allpages> Access public

`string public function Scripto_Service_MediaWiki::getEditToken($title)`

- Function Parameters:
  - `string $title`
- Gets the edit token for a given page.
- Link <http://www.mediawiki.org/wiki/API>Edit#Token>

`array public function Scripto_Service_MediaWiki:: getInfo($titles, [$params = array()])`

- Function Parameters:
  - `string $titles`
  - `array $params`
- Gets basic page information.
- Link [http://www.mediawiki.org/wiki/API:Properties#info\\_.2F\\_in](http://www.mediawiki.org/wiki/API:Properties#info_.2F_in)
- Usedby `Scripto_Document::_getPageInfo()`

`string|null public function Scripto_Service_MediaWiki::getLatestRevisionHtml($title)`

- Function Parameters:
  - `string $title`
- Gets the HTML of the latest revision of a given page.
- Link [http://www.mediawiki.org/wiki/API:Parsing\\_wikitext#parse](http://www.mediawiki.org/wiki/API:Parsing_wikitext#parse)
- Usedby `Scripto_Document::getTalkPagePlainText()`
- Usedby `Scripto_Document::getTranscriptionPagePlainText()`
- Usedby `Scripto_Document::getTranscriptionPageHtml()`
- Usedby `Scripto_Document::getTalkPageHtml()`

`string|null public function Scripto_Service_MediaWiki::getLatestRevisionWikitext($title)`

- Function Parameters:
  - `string $title`
- Gets the wikitext of the latest revision of a given page.
- Link [http://www.mediawiki.org/wiki/API:Properties#revisions\\_.2F\\_rv](http://www.mediawiki.org/wiki/API:Properties#revisions_.2F_rv)
- Usedby `Scripto_Document::getTalkPageWikitext()`

- Used by Scripto\_Document::getTranscriptionPageWikitext()

```
array public function Scripto_Service_MediaWiki::getPageProtections($title)
```

- Function Parameters:
  - string \$title
- Gets the protections for a given page.
- Link [http://www.mediawiki.org/wiki/API:Properties#info\\_.2F\\_in](http://www.mediawiki.org/wiki/API:Properties#info_.2F_in) Access public

```
string public function Scripto_Service_MediaWiki::getPreview($text)
```

- Function Parameters:
  - string \$text
- Get the HTML preview of the given text.
- Link [http://www.mediawiki.org/wiki/API:Parsing\\_wikitext#parse](http://www.mediawiki.org/wiki/API:Parsing_wikitext#parse)

```
string public function Scripto_Service_MediaWiki::getProtectToken($title)
```

- Function Parameters:
  - string \$title
- Gets the protect token for a given page.
- Link <http://www.mediawiki.org/wiki/API:Protect#Token>

```
array public function Scripto_Service_MediaWiki::getRecentChanges([&$params = array()])
```

- Function Parameters:
  - array \$params
- Gets all recent changes to the wiki.
- Link <http://www.mediawiki.org/wiki/API:Recentchanges> Access public

```
string public function Scripto_Service_MediaWiki::getRevisionDiff($fromRevisionId, [&$toRevisionId = 'prev'], $from, $to)
```

- Function Parameters:
  - int \$from The revision ID to diff.
  - int|string \$to The revision to diff to: use the revision ID, prev, next, or cur.
  - \$fromRevisionId
  - \$toRevisionId
- Gets the difference between two revisions.
- Return The API returns preformatted table rows without a wrapping <table>. Presumably this is so implementers can wrap a custom <table>.

```
string public function Scripto_Service_MediaWiki::getRevisionHtml($revisionId)
```

- Function Parameters:

- int \$revisionId
- Gets the HTML of a specified revision of a given page.

```
array public function Scripto_Service_MediaWiki::getRevisions($titles, [$params = array()])
```

- Function Parameters:
  - string \$titles
  - array \$params
- Gets revisions for a given page.
- Link [http://www.mediawiki.org/wiki/API:Properties#revisions\\_.2F\\_rv](http://www.mediawiki.org/wiki/API:Properties#revisions_.2F_rv)
- Usedby Scripto\_Document::\_getPageHistory()

```
array public function Scripto_Service_MediaWiki::getSiteInfo([$siprop = 'general'])
```

- Function Parameters:
  - string \$siprop
- Gets overall site information.
- Link [http://www.mediawiki.org/wiki/API:Meta#siteinfo\\_.2F\\_si](http://www.mediawiki.org/wiki/API:Meta#siteinfo_.2F_si)
- Usedby Scripto\_Document::\_getPageMediawikiUrl()

```
array public function Scripto_Service_MediaWiki::getUserContributions($ucuser, [$params = array()])
```

- Function Parameters:
  - string \$ucuser
  - array \$params
- Gets a list of contributions made by a given user.
- Link <http://www.mediawiki.org/wiki/API:Usercontribs>

```
array public function Scripto_Service_MediaWiki::getUserInfo([$uiprop = "])
```

- Function Parameters:
  - string \$uiprop
- Gets information about the current user.
- Link [http://www.mediawiki.org/wiki/API:Meta#userinfo\\_.2F\\_ui](http://www.mediawiki.org/wiki/API:Meta#userinfo_.2F_ui)
- Usedby Scripto\_Document::\_canEdit()

```
array public function Scripto_Service_MediaWiki::getWatchlist([$params = array()])
```

- Function Parameters:
  - array \$params
- Gets a list of pages on the current user's watchlist.
- Link <http://www.mediawiki.org/wiki/API:Watchlist>

```
bool static public function Scripto_Service_MediaWiki::isValidApiUrl($apiUrl)
```

- Function Parameters:
  - string \$apiUrl
- Determine whether the provided MediaWiki API URL is valid.

```
void public function Scripto_Service_MediaWiki::login($lgname, $lgpassword)
```

- Function Parameters:
  - string \$lgname
  - string \$lgpassword
- Login to MediaWiki.
- Link <http://www.mediawiki.org/wiki/API:Login>

```
void public function Scripto_Service_MediaWiki::logout()
```

- Logout of MediaWiki.
- Link <http://www.mediawiki.org/wiki/API:Logout>

```
bool public function Scripto_Service_MediaWiki::pageCreated($title)
```

- Function Parameters:
  - string \$title
- Returns whether a given page is created.
- Link [http://www.mediawiki.org/wiki/API:Query#Missing\\_and\\_invalid\\_titles](http://www.mediawiki.org/wiki/API:Query#Missing_and_invalid_titles)

```
array public function Scripto_Service_MediaWiki::parse([$params = array()])
```

- Function Parameters:
  - array \$params
- Returns parsed wikitext.
- Link [http://www.mediawiki.org/wiki/API:Parsing\\_wikitext#parse](http://www.mediawiki.org/wiki/API:Parsing_wikitext#parse)

```
array public function Scripto_Service_MediaWiki::protect($title, $protections, [$protecttoken = null], [$params = array()], $protecttokens)
```

- Function Parameters:
  - string \$title
  - string \$protections
  - string|null \$protecttokens
  - array \$params
  - \$protecttoken
- Applies protections to a given page.
- Link <http://www.mediawiki.org/wiki/API:Protect>
- Used by `Scripto_Document::_unprotectPage()`
- Used by `Scripto_Document::_protectPage()`

```
array public function Scripto_Service_MediaWiki::query([$params = array()])
```

- Function Parameters:
  - array \$params
- Returns data.
- Link <http://www.mediawiki.org/wiki/API:Query>

```
array public function Scripto_Service_MediaWiki::watch($title, [$params = array()])
```

- Function Parameters:
  - string \$title
  - array \$params
- Watch or unwatch pages.
- Link <http://www.mediawiki.org/wiki/API:Watch>
- Usedby Scripto\_Document::unwatchPage()
- Usedby Scripto\_Document::watchPage()

```
array protected function Scripto_Service_MediaWiki::_request($action, [$params = array()])
```

- Function Parameters:
  - string \$action
  - array \$params
- Makes a MediaWiki API request and returns the response.